Additional CDAT Packages: Climate Data Specific Utilities: The cdutil Package

cdutil - overview

- The cdutil Package contains a collection of sub-packages useful to deal with Climate Data
- Sub-components are:
  - **times**: a collection of tools to deal with the time dimension, mostly seasonal average
  - **region**: a “region” selector for rectilinear grids
  - **vertical**: already seen earlier in the course
  - **averager**: already dealt with earlier in the course
  - **continents_fill** # Emulate a VCS graphic method to display filled continents
  - **VariableConditioner** and **VariablesMatcher**: A superset of the regridder and time extraction tools, see CDAT doc.

cdutil - “times” module (1)

- The **cdutil.times** module allow the user to manipulate time axes easily, and is mostly geared toward seasonal extraction.
- All seasonal extractions in this module are based on “bounds”, therefore a few function have been created to allow the user to reset the bounds correctly (remember that by default the “time” axis doesn’t have any bounds unless they are in the file).
- These functions are:

  cdutil.times.setTimeBoundsMonthly(slab/axis)

  cdutil.times.setTimeBoundsYearly(slab/axis)

  cdutil.times.setTimeBoundsDaily(slab/axis,frequency=)

cdutil - “times” module (2)

- Once the time bounds are set correctly you can use one of the pre-defined seasonal extractor, or create your own.

- Predefined seasonal extractor are:
  - JAN/FEB/…/DEC <--> ANNUALCYCLE
  - DJF/MAM/JJA/SON <--> SEASONALCYCLE
  - YEAR

- Creating your own:

```
JFMAM=cdutil.times.Seasons('JFMAM')
```

cdutil – “times” module (3)

- Once you created your season extractor, using it is fairly easy:

```
djf=cdutil.times.DJF(slab)
# will return all DJFs computable in your slab
```

- Additional arguments can be passed, the default needs 50% of the season to be present order to assign a value

cdutil – “times” module (4)

- In addition, season extractors have 2 functions:
  - climatology: which basically computes the average of all seasons passed, therefore, ANNUALCYCLE.climatology, will return the 12 month annual cycle for the slab:

    ```
    Ac=cdutil.times.ANNUALCYCLE.climatology(slab)
    ```

  - departures: which given an optional climatology will compute seasonal departures from it.

    ```
    Dep=cdutil.times.ANNUALCYCLE.departures(slab,Ac)
    # Note that since Ac has been computed over the entire time period of slab,
        passing Ac here was superfluous
    ```

- See the documentation for more information:

  http://esg.llnl.gov/cdat/cdat_utilities/cdat_utilities-1.htm#pgfId-998297

cdutil - “region” module

- The **cdutil.region** module allows the user to extract a region “exactly”. i.e. resetting the latitude and longitude bounds to match the area “exactly”, therefore computing an “exact” average when passed to the averager function.

- Predefined regions are:
  - AntarcticZone, AAZ (South of latitude 66.6S)
  - ArcticZone, AZ (North of latitude 66.6N)
  - NorthernHemisphere, NH # useful for dataset with latitude crossing the equator
  - SouthernHemisphere, SH
  - Tropics (latitudes band: 23.4S, 23.4N)

cdutil - region” module

- Creating your selector:

```
myselector=cdutil.region.domain(latitude=(lat1,lat2), longitude=(lon1,lon2)) # can be any dimension, but very useful for lat/lon
```

- Using the selector:

```
slab2=slab1(Myselector)
```

cdutil – “continent_fill” module

The **continents_fill** module emulates a VCS graphic method to display filled contents.

There are 4 steps:

1. Plot your normal plot
2. Create your cf “method”
3. Set cf attributes:

   Fill, fill_color, line, line_color, line_width, projection
4. Plot using the cf, plot method, passing x:

   ```
   cf.plot(x=None,template=None,bg=0,ratio=None)
   ```

Additional CDAT Packages: General Utilities : The genutil Package

General Utilities : genutil

- The “genutil” Package contains tools not necessarily specific for Climate Data.
- Sub-components are:
  - statistics
  - arrayindexing
  - grower
  - picker
  - minmax
  - udunits
  - filters
  - statusbar
  - color

genutil - “statistics”

- The statistic module provides the user with some basic statistics function:
  - (auto)correlation
  - (auto)covariance
  - geometricmean
  - laggedcorrelation
  - laggedcovariance
  - linearregression
  - meanabsdiff
  - median
  - rank
  - rms
  - std
  - variance
- CDAT documentation and doc string develop these:

```
>>> help(genutil.statistics.geometricmean)
```

genutil - “statistics” – correlation (1)

- **genutil.statistics.correlation()** returns the correlation between 2 slabs. By default on the first dimension, centered and biased by default.
- Slabs must be of the same shape and size.

Usage:

```
result = correlation(slab1, slab2,
   weights=weightoptions, axis=axisoptions,
   centered=centeredoptions,
   biased=biasedoptions)
```

Options:

weightoptions

default = None. If you want to compute the weighted correlation, provide the weights here.

NOTE: the weights array must be the same shape and size as the slabs.

genutil - “statistics” – correlation (2)

Options (continued):

axisoptions ‘x’ | ‘y’ | ‘z’ | ‘t’ | ‘(dimension_name)’ | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered.

biasedoptions None | 0 | 1

default value = 1 returns biased statistic. If want to compute an unbiased statistic pass anything but 1.

genutil - “statistics” – correlation (3)

Example:

```
>>> import cdms, genutil
>>> f=cdms.open(‘file1.nc’)
>>> var1=f(‘u_wind’)
>>> var2=f(‘v_wind’)
>>> corr=genutil.statistics.correlation(var1, var2,
  axis=“tzyx”)
>>> print corr
correlation
array(-0.237745400348)
```

genutil - “statistics” – std (1)

- **genutil.statistics.std()** returns the standard deviation from a slab. By default on first dimension, centered, and biased.

Usage:

```
result = std(slab, weights=weightoptions, axis =
   axisoptions, centered=centeredoptions, biased
   = biasedoptions)
```

Options:

weightoptions

default = None. If you want to compute the weighted correlation, provide the weights here.

NOTE: the weights array must be the same shape and size as the slab.

genutil - “statistics” – std (2)

Options (continued):

axisoptions ‘x’ | ‘y’ | ‘z’ | ‘t’ | ‘(dimension_name)’ | 0 | 1 ... | n

default value = 0. You can pass the name of the dimension or index (integer value 0...n) over which you want to compute the statistic.

centeredoptions None | 0 | 1

default value = 1 removes the mean first. Set to 0 or None for uncentered.

biasedoptions None | 0 | 1

default value = 1 returns biased statistic. If want to compute an unbiased statistic pass anything but 1.

genutil - “statistics” – std (3)

Example:

```
>>> import cdms, genutil
>>> f=cdms.open(‘file1.nc’)
>>> var1=f(‘u_wind’)
>>> var1.shape # just a linear variable
(9,)
# We want to set some weights to add importance to
# the higher range of values
>>> wghts=[0.2, 0.3, 0.5, 0.6, 1.0, 1.0, 1.2, 1.4,
           1.0]
>>> std=genutil.statistics.std(var1, weights=wghts)
>>> print std
0.7340166556835906 5
```

genutil – “arrayindexing” (1)

- “**arrayindexing**” allows the user to do exactly what its name says, setting or retrieving parts of a slab, not via a single array, but via an array representing which index to get at each cell. i.e:

```
C=Array[Indices] # where Indices is an array
```

genutil – “arrayindexing” (2)

Example:

```
>>> A=MV.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
>>> I=MV.array([0,0,0,0])
>>> genutil.arrayindexing.get(A,I)
variable_21
array([1,2,3,])
>>> A[0]
variable_21
array([1,2,3,])

>>> A[1]
variable_21
array([4,5,6,])

>>> I=MV.array([1,0,1,0])
>>> genutil.arrayindexing.get(A,I)
variable_21
array([4,5,6,])
```

The “grower” function

>>> help(genutil.grower)
Help on function grower:

grower(x, y, singleton=0)
Function: grower

Description of function:
This function takes 2 transient variables and grows them to match their axes.

Usage:
x, y = grower(x, y, singleton=singletonoption)

Options:
singletonoption 0 | 1
Default = 0 If singletonoption is set to 1 then an error is raised if one of the dims is not a singleton dimension.

genutil - “picker” (1)

- “picker” allows to select non contiguous values of an axis, for example:

```
mypick=genutil.picker(level=(100,850,200))
s2=s(mypick)
```

- An additional “match” keyword can be provided to the picker.

- If “match” is set to 1 then all requested values must be present, if set to 0 then non-existent values will be returned with “missing_value” everywhere, if set to -1, then non-existent requested values will be skipped.

genutil - “picker” (2)

- This **picker** example shows how you can do strange things to your variable very quickly and easily. Suppose you wanted to select a discrete number of latitudes (10°N, 43°N, 86°N and 90°N):

```
>>> import cdms, genutil
>>> var=cdms.open('myfile.nc')('myvariable')
>>> mypick=genutil.picker(latitude=(10, 43, 86, 90))
>>> newvar=var(mypick)
>>> print newvar.getLatitude()[:]
[ 10., 43., 86., 90.,]
```

genutil - “udunits”

- The “**udunits**” module is a python port of the C/Fortran “udunits” conversion package:

```
>>> from genutil import udunits
>>> print udunits.__doc__ # OR   help(udunits)
>>> from genutil import udunits
>>> myunit=udunits(5.6, "m s**-1")
>>> myunit.to("knot")
udunits(10.8855291577,"knot")
```

- To search units for keyword “meter”:

```
>>> for unit in myunit.known_units():
...     if unit.find("meter")>-1:
...         print unit
```

genutil - “minmax” and ”filter”

- The “**minmax**” function returns the minimum and maximum values of anything that is passed to it, you can use mixed types. Note that vcs has its own minmax, the difference is that the “vcs” version masks everything greater than 1.E20 (in absolute value).

- The “**filter**” module is in its early stage and contains at this point very minimal filtering capabilities: “running average”, “self build weights based filters” and “121 smooth” filter.

genutil - “colors” and ”statusbar”

- The “colors” module is identical to that in **vcs**.

- The statusbar module allows for either GUI or prompt based “statusbar” telling to inform the user when some processing is taking place.
  - Example:

```
prev=0
for I in range(1000):
  prev=genutil.statusbar(I,total=1000,
                         title='Status:',prev=prev)
```

Status: ##| 99.90%>>>